

CS51 Assignment 0: The Best Laid Scheme

Due: Friday, 6 February 2009 at 5:00 PM

Total Points: 40

In this assignment you will set up your CS51 environment and learn the basics of Scheme. Reference chapters: How To Design Programs, Chs. 2, 3, 4, 9. (See the website for a link to the textbook, or follow the following link: [http://www.htdp.org/2003-09-26/Book/.](http://www.htdp.org/2003-09-26/Book/))

1 Sectioning and bulletin board

(2 points total)

Exercise 1.

2 points

Your first task is to sign up for a section. Sectioning will open on Sunday, Feb. 1, and will end when this assignment is due on Friday, 6 February 2009 at 5:00 PM. When sectioning opens, you will be able to read about how to section here:

<https://www.section.fas.harvard.edu/>, with detailed instructions at:

http://www.registrar.fas.harvard.edu/fasro/sectioning/instructions_student.jsp.

(**Note 1:** The link above will not be active until sectioning opens. **Note 2:** You must section by Friday, 6 February 2009 at 5:00 PM, even if you plan to use late days on the assignment.)

Your second task is to sign up for an account on the course bulletin board and *read the bulletin board guidelines*. You can find the link to the CS51 BB on the CS51 website; click the “Bulletin Board” link on the left-hand side of the page, or go to <http://cs51.seas.harvard.edu/phpBB3>. As in CS50, the CS51 staff will be able to see your name when you post, but other students will not, so you still have anonymity.

If you run into any issues on the problem set, please do not hesitate to post your questions in a new topic under the Problem Set 0 section of the Bulletin Board! A TF will answer any questions you post quickly.

2 Good style

(5 points total)

Style is one of the most elusive parts of a computer science problem set, but in this course it will also be one of the most important (remember - we aim to write beautiful code). Detailed style guidelines can be found on the website at <http://cs51.seas.harvard.edu/docs/style.pdf> (there is also a permanent link on the Course Policies page). You should read and review these very carefully! Every assignment will have some number of style points associated with it; this one has five. I will also remind you that as stated in our Course Policies, all requests for regrades need to go to the Head TFs, who will regrade the entire problem set from scratch. (Thus, your grade may go up or down.)

One important thing to note is that we are requiring that your submission be at most 79 characters per line. This is a standard restriction for code; many text editors will automatically default to an 80-character line length, meaning that longer lines will be wrapped, which makes your code harder to read and understand.

Exercise 2.

5 points

Use good style throughout your assignment!

3 Getting started

3.1 Submission/Retrieval System: Explanation

This semester, we will be working almost entirely from within DrScheme. DrScheme is an IDE (Integrated Development Environment) for Scheme, which basically means that it's software intended to allow you to develop Scheme more easily. However, DrScheme has many more advantages than any IDE that most languages have available, to the extent that much of what we will do with Scheme will be tied directly to functionality that DrScheme provides.

The first example of this is the way that we will be doing problem set distribution and submission. All problem sets will be posted in PDF format on the appropriate section of the website as they become available; you are probably reading this right now from the PDF of PS0. However, for most assignments we will give you some kind of code skeleton to fill in. You will typically retrieve this code skeleton directly from within DrScheme. You will then modify it as necessary to complete the assignment and submit your modified version in the same way.

We will sometimes exert certain controls over your submission. Most commonly we will require your submission pass certain tests before we accept it. If your submission fails one of our required tests, you will receive a message informing you that your code doesn't work (this message might be more or less descriptive depending on how helpful we're feeling that week), and you will not be able to submit until you've identified and fixed your bug. This means that testing will be extremely important! Nothing is more irritating than realizing

that your code has a bug and having to start writing tests from scratch, because you will have no idea how to go about isolating the problem. Testing is something that we will be emphasizing throughout the course.

3.2 Setting Up DrScheme

Now then: to the actual setup. The DrScheme software is part of a larger project called PLT Scheme.

Note about the computer lab: We are in the process of installing the version of Dr. Scheme required for this course onto the Linux computers in the computer lab. There will be a course announcement about how to work on these computers when we get them working, but until then please use your own. If this is a problem, contact the head TFs.

1. Point your web browser at: <http://download.plt-scheme.org>. Select your platform (it should auto-select it for you) and click Download. Any of the mirrors is fine.
 - If you are on a PC, you will be downloading an installer. When you've finished the download, run the installer; all of the default options are fine, so you can just click through. It will perform the installation for you and automatically start up DrScheme.
 - If you are on a Mac, you will be downloading a disk image. Once it's done downloading, double-click the disk image to mount it; it will contain one folder called PLT Scheme v4.1.4. Drag this folder to wherever you'd like it to live on your computer, then unmount the image by ejecting it. (Some Macs will automatically mount the disk image, put the PLT Scheme folder somewhere on your computer, and then unmount the disk image for you; if it seems like the download is taking a long time, this is probably what's going on.)
 - If you run something other than Windows or OSX, you can probably figure it out for yourself. But if you are having any trouble, post your situation to the bulletin board and we will do our best to help.
2. You should now start up DrScheme from wherever you saved it, if you haven't already. The first thing you should do is select your language. For this assignment, you should select Languages – Choose Language..., expand the “How to Design Programs” menu option, and select “Intermediate Student”. What this means is that we will be using a subset of real Scheme for now; it won't allow you to do certain more complicated things that could trip you up until we talk about them in class. Then click OK.
3. Now select File – Install .plt File..., and when you are prompted for a URL (make sure the Web tab is selected), type in: <http://cs51.seas.harvard.edu/cs51.plt>. It should perform some setup work, which might take up to a few minutes. When it's done, close the setup dialog box. Then quit and restart DrScheme.

3.3 Creating a Submission Account

You're getting there! At this point, you should have a button at the top of your DrScheme window called "CS51 Handin".

4. But, you must first create a handin account. This account will be what you use for all future homeworks in the course, so please fill out the information correctly, or there may be confusion about assigning you your grades. Select File – Manage CS51 Handin Account... and provide all of the information requested. Your Username should be your FAS email login - this is important! Please provide your full name (e.g., Gideon Wald) and your fas email in the appropriate spaces. Your ID number should be your HUID. Choose whatever password you want - you won't have to share this password with anyone this semester. Then click Add User and wait until it's finished before exiting the dialog box.
5. You are now ready to begin work! The magic button that lets you do all the stuff we've been talking about is the "CS51 Handin" button. It allows you to submit the code in the DrScheme window to our handin server automatically, which is pretty nifty. But - at least as cool - it also lets you retrieve your most recent submission. (This means that you will always have a remote copy of your most recent submission saved; it also means that you can load up DrScheme on any computer and immediately retrieve your work.) Normally, we will pre-load any code skeletons you need into your submission directory, but we couldn't do that for this assignment since you just created your account. So, just for this week, download `asst0.scm` from the website and open it in Dr. Scheme to begin your work. You can find the link on the homepage, or visit <http://cs51.seas.harvard.edu/docs/hw00.scm>.

If you run into trouble with any of the above, please e-mail the Head TFs, Gideon and Victor (gwald@fas and shnyder@gmail).

Please also note that in the future, we might be making changes to the backend of this system that will require you to download an update. Don't worry; this is super easy, and, in fact, you will be automatically prompted to download any updates immediately when you start up DrScheme, if there are any available. If this happens, just click "Update Now", wait for the setup to finish, and **restart DrScheme**.

3.4 What you must do

If you've followed directions above, you should have a DrScheme window open containing the code you need to complete the assignment. This window contains written questions, programming exercises, and a few tests. You should answer the questions and complete the exercises in that file. Any exercises that require you to answer a question in words/complete sentences rather than code should be written into your Scheme file as a **comment** (comments in Scheme start with a semicolon); any exercises that require you to write code should be written as **code**, including variable definitions. (We've commented out the partial

declarations with “YOUR-ANSWER-HERE” in place of a function definition so that you can immediately Run when you download the file, but you should uncomment these when you’ve completed each exercise.)

When you tell DrScheme to run the code (click the “Run” button at the top), much of your code will be highlighted in black. This is an extremely convenient feature of the Intermediate Student language level of DrScheme: it tells you which branches of your code you are not testing. In other words, if there is a command in your .scm file that exercises a certain branch of code, it will un-highlight it. Note that having none of your code be highlighted is a necessary but insufficient condition for being done with testing. You must test every branch of your code before you can be sure that you’ve checked every case, but the fact that all of your code has been run does not mean that you haven’t forgotten a branch that should be there! You can also type commands at the interpreter to see which ones exercise which branches of code.

You can save it locally on your computer however you like; however, it is probably more convenient to periodically upload your work to the CS51 server using the Handin tool. When you click the “CS51 Handin” button at the top of your DrScheme window, you will be asked for your username and password, and then you will need to select an assignment folder. “scratch” will be an option on every assignment; it is simply a scratch space, where you can store the most recent version of your code for your own use. hw00 will be where you will submit your final version; it’s subject to some testing controls (more details to follow). The questions and exercises are also printed below for your convenience.

3.5 Submission - please read carefully!

As explained above, the really amazing thing about this setup is that it allows you to submit directly from within DrScheme. If you’ve followed the instructions in the “Getting Started” section correctly, you should be able to use the “CS51 Handin” button at the top of your DrScheme window for submission. Make sure that when you want to submit your work, you don’t have the “Retrieve” button checked, and that you are using your own account. (Note that with the “Retrieve” button checked, the button changes from “CS51 Handin” to “CS51 Retrieve”.)

Only your last submission of a given problem set gets saved. Your most recent submission gets graded, and if it is sent after the due date, it will be considered late—even if you submitted an earlier version of your work before the deadline. **This last submission should be submitted to the “hw00” Assignment, NOT the “scratch” assignment; all submissions in the scratch space will be considered for your use only.**

This homework assignment is due on Friday, 6 February 2009 at 5:00 PM. Homework submitted after this time will cost you late days.

The rules for submission are specified in more detail in the course policy document.

4 Scheme questions

(7 points total)

Exercise 3.

2 points

(a) [2 points]

Tell us about yourself! In `asst0.scm` we have defined five string variables. Replace the values of these variables with the answers to the following questions.

- What is your past CS experience?
- What other interests do you have?
- What is your (prospective) concentration?
- What do you hope to get out of this course?
- On a scale of 0 to 2, 2 being the most confident, how sure are you that you are taking this course?

Exercise 4.

5 points

Write the following expressions and values as Scheme expressions. You may wish to check your work in the runtime interpreter. For example, the expression “1 + 2” would be `(+ 1 2)`, and the value “A list containing 1 and 2” would be `(list 1 2)`.

(a) [1 point] `1 + (2 * 3 / 4) - (5 - 6)`

(b) [1 point] A list containing the integers from 1 to 5 in ascending order.

You’ve seen `car` (gets the first element of a list) and `cdr` (gets the rest of the list) in class. But what if you want to extract the second element? (Note that this is not the same as getting the “rest” of the list; why?) In this case, one might define a function named `cadr`, that is, the `car` of the `cdr`. `caddr` would be the `car` of the `cdr` of the `cdr`, i.e. the third element; etc. These functions (and related ones) do in fact already exist in Scheme.

(c) [1 point] Implement `my-cadr`, your own version of the built in function `cadr`. Obviously, you may not use the function `cadr` in your definition - that would be cheating!

```
> (my-cadr (list 1 2 3))
2
> (my-cadr (list 4 (list 1 2) 3 9))
(list 1 2)
```

The variable `lst` is defined below. For the following two questions, extract the requested values from `lst` using `car`, `cdr`, and any other variants (e.g., `cadar`, `caddr`). (Note: for this part you may use the built-in versions of these functions.)

```
(define lst (list 1 (list 2 3) (list #f "cs51") 4))
```

(d) [1 point] Get the second element of `lst`.

(e) [1 point] Extract the value `#f` from `lst`.

5 Programming exercises

(26 points total)

Don't show too much surprise! For most Scheme exercises in this course, the use of functions ending in `!` (such as `set-cdr!`) is not permitted. We will explicitly tell you if an assignment isn't subject to this restriction.

Exercise 5.

8 points

A predicate is a function that returns `#t` or `#f`. For example, `odd?` is a predicate that returns `#t` if its argument is an odd integer and `#f` if its argument is an even integer. In Scheme, it is traditional to give predicates names ending in a question mark.

(a) [3 points] Define a predicate (`inside-unit-circle? x y`) that returns `#t` if the coordinate (x, y) is within the circle of radius one centered at $(0, 0)$ and `#f` otherwise. (If a point lies on the unit circle, it should be considered within it for the purposes of this question.) Remember, a point is inside the unit circle if the distance of (x, y) from origin, $(0, 0)$, is less than 1. The built-in scheme function `sqrt` may be helpful, although you don't actually need it (why not?). Your solution must handle all inputs, including non-number inputs.

For example:

```
> (inside-unit-circle? 1 0)
#t
> (inside-unit-circle? .5 .4)
#t
> (inside-unit-circle? 1 2)
#f
> (inside-unit-circle? .5 'zero')
#f
```

(b) [2 points] Define a predicate (`outside-unit-interval? x`) that checks whether an input value `x` is a number outside of the range $[0, 1]$ (inclusive - i.e., 0 and 1 are within this range, and thus (`outside-unit-interval? 0`) should evaluate to `#f`). Note that this predicate should only return `#t` if the input is a number.

Examples:

```
> (outside-unit-interval? .5)
#f
> (outside-unit-interval? 1)
#f
```

```
#f
> (outside-unit-interval? -1)
#t
> (outside-unit-interval? "1")
#f
```

(c) [3 points] Define a function (`filter-outside-interval lst`) that takes as input a list of values and returns a list of only those elements that are numbers outside of the range [0, 1]. Hint: you might find the predicate you just wrote useful in this section.

Examples:

```
> (filter-outside-interval (list 0 3 .5 -2 "greg"))
(list 3 -2)
> (filter-outside-interval (list))
empty
> (filter-outside-interval (list (list 0 1) 15 -50))
(list 15 -50)
```

Exercise 6.

9 points

(a) [4 points] Define a function `num-occurs` that takes an element and a list as input and returns the number of times that the element occurs in that list. Use the function `equal?` to check for equality.

```
> (num-occurs 1 (list 1 2 3 1 2 3 1))
3
> (num-occurs 1 (list 1 2 3 4))
1
> (num-occurs 5 (list 1 2 3 4 (list 5 6)))
0
```

(b) [1 point] In the last example input above, why should the function return 0, even though it seems that 5 is in the list `(list 1 2 3 4 (list 5 6))`?

(c) [4 points] Write `num-occurs-deep`, a modified version of `num-occurs` that will return the number of times an element occurs anywhere in the list, including within nested lists.

For example:

```
> (num-occurs-deep 1 (list 1 2 3 1 2 3 1))
3
> (num-occurs-deep 5 (list 1 2 3 4 (list 5 6)))
1
> (num-occurs-deep 0 (list (list (list 0 0))))
2
```

Exercise 7.

4 points

As explained in the "Getting Started" section, we will sometimes have tests that your assignment must pass in order for your submission to succeed. For this assignment, the tests we've given you in your Scheme file are the ones that we will be **requiring** your program to pass. However, there are a couple of important cases that we're missing (some of which we've hinted at in the comments above...). In particular, what other types of input should be tested for `inside-unit-circle?`, `outside-unit-interval?`, and `filter-outside-interval?` (Hint: see our examples.) What about `num-occurs` and `num-occurs-deep?` (Hint: on what types of inputs should we see these two functions behave differently?)

(a) [4 points] Please write any additional tests you feel you need in order to ensure that your functions work correctly in all cases.

There is a built-in function in the Intermediate Student language called (`check-expect` a b). It takes two arguments and checks that they are equal. So, you should provide it your function call and the value you expect it to take on. If the tests pass, it will print a message telling you that they all passed; otherwise, a helpful window will pop up informing you of which tests failed and what the actual values ended up being.

Example:

```
> (check-expect 1 1)
> (check-expect (cadr (list 1 2 3)) 2)
> (check-expect (+ 2 2) 4)
All three tests passed!
> (check-expect 1 9)
Actual value 1 differs from expected value, 9.
```

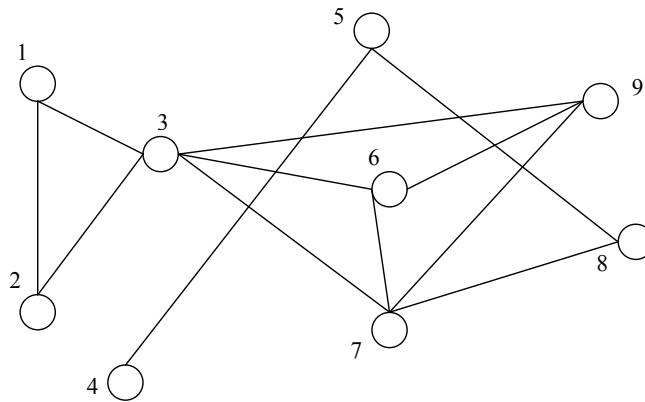
Exercise 8.

4 points

In the graph below, nodes are labeled with numbers. Recall from lecture that a clique is any group of nodes, all of which are connected to each other node in the group. Find all cliques of size three or greater. Represent your answer as a variable `all-cliques` which should be defined as a list of lists, where each sub-list (of size three or longer) represents a clique. List nodes in ascending order.

Note that when we ask you for all cliques, we want you to include cliques which are sub-cliques of other cliques. Thus, if nodes a, b, c and d are all connected to each other, that implies that the four sub-cliques (a, b, c), (a, b, d), (a, c, d), and (b, c, d) are also cliques of size three or greater. You should list all of them.

Example: `(define all-cliques (list (list 1 3 4) (list 1 6 7 9) (list 4 7 8)))`

**Exercise 9.***1 point*

Please tell us how much time you spent on this problem set! The way we'd like you to do this is by defining a variable `minutes-spent` to be equal to the approximate number of minutes you spent working on this, from start to finish. Please don't forget this, or we'll think you finished in negative time, and students that fast clearly need more work. (But in seriousness, be honest - this is confidential and important for us to tune future problem sets.)